J Netw Syst Manage (2017) 25:884–901
DOI 10.1007/s10922-017-9437-9

Challenges and Directions in Service Management Automation

Alexander Keller[1]

Received: 8 August 2017 / Revised: 25 September 2017 / Accepted: 3 October 2017 /
Published online: 25 October 2017

Abstract Research teams and IT service providers are continuously increasing the degree of Automation in Hybrid IT environments spanning multi-provider Clouds as well as traditional IT. Automation and Cognitive Systems are widely regarded as the foundation for improving the productivity as well as the quality of Service Delivery. At the same time, Hybrid IT deployments are being subject to a variety of challenges in large deployments on a global scale. Based on our experience of running a service practice that delivers IT Service Management in Hybrid IT environments to customers worldwide, this paper will review the challenges and directions in implementing Integrated Network, Systems and Service Management technologies and point out gaps encountered in current tools and implementations. This is fueled by the desire of Service Providers to evolve Automation from a development activity to an approach focusing on recording and storing the practices of the best system administrators and further on to a cognitive activity where the management system learns over time which actions need to be undertaken under a given set of circumstances. By means of real-life examples, we identify candidate concepts and technologies that will help us build the next generation of Integrated Network, Systems and Service Management.

Keywords Service Management · ITIL · Best Practices · Automation · Cloud Computing · Hybrid IT

✉ Alexander Keller
alexk@us.ibm.com

[1] IBM Global Technology Services, 71 South Wacker Drive 7th Floor, Chicago, IL 60606, USA

1 Introduction

The 25 year milestone of *Journal of Network and Systems Management* (*JNSM*) allows us to reflect on the tremendous progress in information technology (IT) in general and the Internet in particular. Evidently, 25 years ago, IP based protocols and a good deal of the core services (domain name service, email, file transfer, news, telnet, whois, management) were commonplace for Academics and Researchers, but technologies integrating these services in order to provide a seamless user experience such as Hypertext, search engines and web browsers were still in their early stages and hence didn't make for widespread adoption by the broader public in the early 90s [1].

While the core services and principles of the early Internet still apply today, the stunning evolution of a vast array of mobile apps has revolutionized the way we live: Each of us carries one or more smartphones and portable computers with us that are equipped with a variety of sensors (gyroscopes, GPS sensors, heart rate sensors etc.). They are always connected, sending data about our status, position, and even health to back-end transaction processing systems, which—in turn—feed information back to us. In the example of traffic and telematics applications depicted in Fig. 1, the GPS and time data from our phone are aggregated in the back-end systems of Google (Fig. 1a) and Waze (Fig. 1b) [2] to calculate current speed, exact beginning and end of traffic slowdowns, impact on the travel time and (a very precisely) estimated time to arrival. In addition, social media-sourced

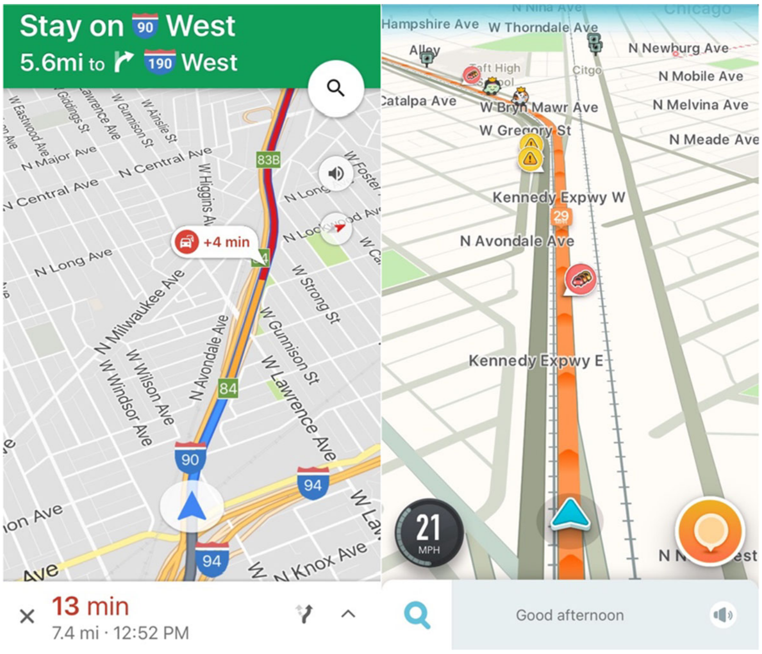


Fig. 1 Ubiquitous instrumentation example: traffic sensors and arrival time prediction

information pertaining to the location of root causes (accidents, stopped vehicles, road hazards, police, speed cameras) as well as hints and tips from other community members (gas prices) are included in the maps. If alternative paths exist, their duration is forecasted, and the driver is made aware of a better alternative for routing its journey.

By doing so, these traffic apps execute the exact same feedback control loop that our IT Operations and Management systems execute: they (1) *monitor* speed and position, (2) *analyze and contextualize* the data into information, and (3) *plan* potential alternatives by forecasting key performance indicators. The (4) *execute* [2] step is left to the driver, although it is foreseeable that a self-driving car would be able to input this information in order to optimize the trip without further human intervention.

We point out that the highways themselves are virtually unchanged from 25 years ago, and the modest attempts at *instrumenting the highway itself* by introducing telematics (often merely a digital panel over the road displaying '15 Minutes to Junction with highway 190') are dwarfed by the user experience introduced *by instrumenting each person sitting in a car* on that same highway. We also note in passing that the above applications are available for free, whereas Network, System and Service Management platforms still run in the hundreds of thousands of dollars even for a medium-sized corporation. It is hence fair to ask why we have not seen similar progress in managing our data centers.

In spite of their high cost, Network and Systems Management platforms are built according to traditional software architecture (usually Java Enterprise Edition). They still use the Simple Network Management Protocol (SNMP) as their 'lingua franca' when managed resources need to be monitored which do not expose their instrumentation via APIs. Hence, it is not surprising that some of the open problems described 25 years ago are still not fully resolved as they usually revolve around Automation in several key areas: First, the automation of provisioning activities, and—once the systems have been set up—the automation of service management processes, such as assessing the impact of a Change Activity. Asset (hardware devices and software licenses) and Configuration Item (infrastructure elements) discovery and reconciliation processes are often incomplete and lead to suboptimal results by leaving the system of record, namely the *Configuration Management DataBase/System* (*CMDB/CMS*) in an inconsistent state, which, in turn, has a ripple effect on chargeback and billing as well as on measuring Service Level Agreements (SLAs). This is also often true in Cloud environments, and especially in Hybrid IT setups that blend Cloud with traditional or virtualized on-premise infrastructure.

The contribution of this paper is to pinpoint several of the aforementioned issues and suggest improvements, based on our experiences at IBM Global Technology Services, a leading worldwide Service Provider for both project based delivery (build management systems and subsequently hand them over to a customer) as well as strategic outsourcing [3] (build systems, transform the operations and subsequently run/manage the distributed infrastructure in steady state). The paper is structured as follows: In Sect. 2, we analyze the inhibitors that made the progress in Service Management move slower than the swift speed of IT in general. We will subsequently describe in Sect. 3 several key areas of innovation in Service

Management that either have or are about to significantly increase the efficiency and effectiveness of Service Management. We conclude the paper in Sect. 4 and present areas for further research.

2 Impediments to Service Management Automation

Service Management Automation as a driver for the Industrialization of IT has been the subject of high hopes and optimistic forecasts: a Gartner report [4] predicted in early 2014 that 'by 2017, managed services offerings leveraging autonomics and cognitive platforms will permanently remove head count to drive a 60% reduction in the cost of services'. There is no question that today, these percentages are only attained under very fortunate circumstances (e.g., rigorously enforced homogeneity of managed resources, special-purpose environments, automated Dev/Test systems), and are nowhere near a 60% reduction today in typical enterprise environments. That said, Software Defined Networks allowing the provisioning of networks on-the-fly, Software Defined Storage solutions performing e.g., the dynamic reclassification of storage tiers reduce the friction of administering large-scale systems and have already contributed to increased efficiencies in systems management.

In order to establish a frame of reference for our discussion, Fig. 2 depicts the major building blocks of a Service Management Automation solution that Service Providers implement for their clients. This is true for both on-premise installations, or Cloud and Software-as-a-Service (SaaS) environments. Each building block represents a major capability of the targeted Service Management environment to be built.

The top layer *Service Management Process Solution* building block shows the process-based bundles of functionality which implement ITIL Best Practices [5].

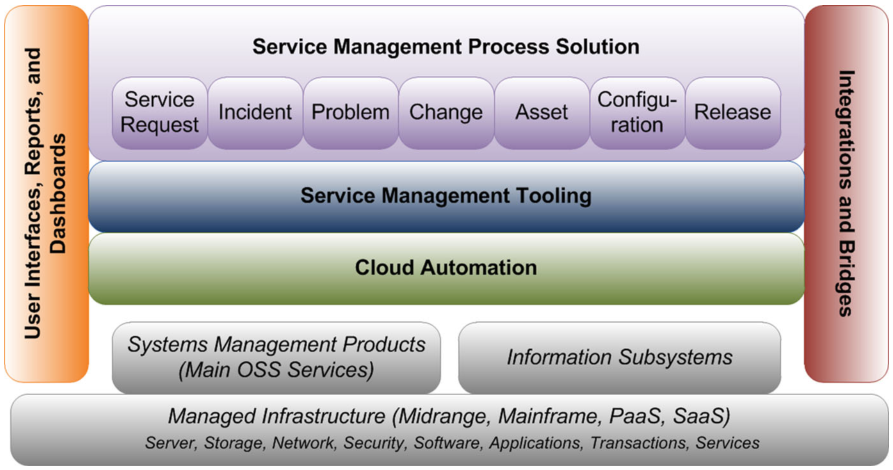

User Interfaces, Reports, and Dashboards
Service Management Process Solution
Service Request
Incident
Problem
Change
Asset
Configu-ration
Release
Service Management Tooling
Cloud Automation
Integrations and Bridges
Systems Management Products (Main OSS Services)
Information Subsystems
Managed Infrastructure (Midrange, Mainframe, PaaS, SaaS)
Server, Storage, Network, Security, Software, Applications, Transactions, Services

Fig. 2 Service management solution building blocks

Each bundle consists of key applications, process implementations, and data model extensions necessary to enable the Service Management processes. These are:

1. Service Request Fulfilment and Service Request Catalog Management
2. Incident Management (including Major Incident Management)
3. Problem Management
4. Change Management
5. Asset Management
6. Configuration Management
7. Release/Deployment Management

The Service Management Process Solution block relies on the *Service Management Tooling* building block, which will provide the infrastructure and toolset required to perform Service Management processes. The Service Management Tooling building block is tightly coupled with the (*Hybrid*) *Cloud Automation* building block. They provide the necessary configurations and procedures in the SM tooling to allow handling of both Cloud and traditional IT services in parallel and preferably using the same tools and automation frameworks (e.g., CHEF, Salt, Puppet). Although automation for Cloud and non-Cloud environments is an integrated part of the Service Management Toolset solution, its implementation is handled in a separate building block to reduce complexity within the Service Management Tooling building block itself. The *Cloud Automation* building block is implemented by typical Cloud Orchestration tools, such as VMWare vRealize, or through integration with public Cloud Management APIs from Amazon Web Services, Microsoft Azure, or IBM Bluemix [6].

Integrations and Bridges is used as a generic block to indicate the various integrations of the Service Management Toolset building block with external systems. The key element of this building block will be an *Enterprise Services Bus (ESB)*: this is the central service integration layer for interfaces between the different IT Service Management tools. The building block additionally includes:

- Setup and configuration of necessary interface adapters in the Service Management toolset
- Integration to Network and Systems Management Products and Information Subsystems that form a part of the Systems Management architecture
- Integration to supporting services such as an LDAP source for authentication of users or a connection to an email service for email notifications.

Building block *User Interfaces, Reports, and Dashboards* enables user access to the Service Management toolset and data. User access to this building block is mainly for service management roles. The building block also includes access to a standard set of reports and preconfigured dashboards that are part of the Service Management environment. In addition, a data warehouse reporting and analytics solution is part of this building block.

Systems (*and Network*) *Management Products* provide the glue between service management and managed infrastructure by implementing the aforementioned *monitor* and *execute* steps of the feedback control loop. They are vital to the success of Service Management Automation.

In the following sections, we will describe the key impediments to the success of Service Management Automation that we have witnessed in the field.

2.1 Poor or Insufficient ITIL Implementations

The IT Infrastructure Library (ITIL) [5] is the incumbent set of best practices for IT Service Management. Over the past years, the rigor of the ITIL framework has led to improved stability of IT Operations and Management; however, it is increasingly viewed as a key contributor to often month-long change and release cycles, and hence bears some responsibility for the delayed introduction of key new features into an IT environment. In contrast, it has long been understood in software development circles that only an iterative, agile development approach can address the need for rapid introduction of new features into a distributed software system. Carrying this concept further into the deployment phase of a software product yields the concept of DevOps (aka Continuous Delivery), where the development, testing and operations teams closely collaborate to release software builds into production in a rapid and incremental fashion. The aforementioned rigidity of the ITIL framework and its associated processes is often viewed as an impediment to the need for rapid, iterative changes in a DevOps world. The key contributing factors are discussed in the following subsections.

2.1.1 Rigor Impeding Flexibility

While ITIL itself only prescribes a specific workflow in very few cases, it is very specific to the necessary tasks, gates and supporting documentation for each activity in an IT management process. For example, ITIL Service Transition [5] separates changes into standard, normal and emergency changes, with the latter two requiring a Change Approval Board. In environments with major code releases on a weekly basis, often coupled with daily drops of additional functionality, fulfilling the formal needs of a normal change is viewed as prohibitive.

IT departments wrongly insisting on strict adherence to a normal change management process for all possible changes find themselves circumvented by ‘Shadow IT’. This, however, is a deliberate (and erroneous) interpretation by the IT department, and not an ITIL issue per se: ITIL clearly states that low-risk, recurring changes are best classified as standard, pre-approved changes. What is happening in the field, however, is that the change management process is often misused by teams to manage the demand and hence their workloads by, e.g., insisting on long lead times [7].

2.1.2 Implementation and Capability Gaps

Until today, some key activities mandated by ITIL are poorly implemented, which yields to the execution of tasks by humans, thus making them inevitably subjective as well as prone to delay and errors. A typical example of a key change management task with sorely lacking system support is assessing the impact of a change, which requires up-front determination as to which elements in the software stacks and ultimately business services could be impacted. The risk assessment is often carried out by a human, who can only rely to some extent on the data within the CMDB/CMS, because discovery technologies continue to be a disappointment in terms of:

1. ability to discover fine-grained software artifacts,
2. supporting the most current versions of a constantly changing middleware, RDBMS and infrastructure component landscape, such as load balancers or appliances,
3. poor scalability of the system, which often leads to scans being performed only once a week, with most discoveries being limited to scanning merely IP address ranges without credentials.

There is broad agreement that this traditional process of filling the CMDB with data (scan → reconcile → transform → promote) spanning multiple tools is inherently broken. Section 3.4 will discuss approaches for mitigating these shortcomings.

Another example for an often-encountered capability gap is the integration of server patching with change management. The development of automated integrations between these disparate systems is usually left to a company's IT organization, or to a service provider. While the latter has usually the economies of scale to build and subsequently amortize the cost of building these integrations by means of frequent reuse, a company merely purchasing these systems will most likely not spend the additional money in integrating them.

2.1.3 *Ticket-Driven Versus Outcome-Driven*

Inflexible forms for reporting issues and requesting help as well as hard-to-navigate Voice Response Units (VRU) have led to considerable frustration with Service Desks among end users. Integrated chat capabilities were viewed as a good middle-ground when they were introduced a few years ago; however, their limited scalability due to a declining number of human operators makes this option increasingly unattractive. Please refer to Sect. 3.2 for a discussion on how to improve the effectiveness of the chat interface.

The quality of community-provided content in a self-help model varies, and service management tools aren't always good at content management, especially when it comes to rich media. This is at least partially due to the fact that few enterprises are willing to incur costs for manually keeping a knowledge management system comprising curated articles and how-to documents up-to-date, especially when equally good content for common-off-the-shelf products can be found for free on the Internet.

In the current state of the art, the concept of a ticket is forced upon a user submitting an incident or service request, and ticket closure rate targets often yield to suboptimal quality (ticket closed → problem solved). Customer satisfaction follow-up by means of emails contributes to a very low return rate of feedback, and usually captures negative responses.

2.2 Legacy Architectures Impeding Continuous Delivery

Initial enthusiasm for Continuous Delivery/DevOps technologies suggested that traditional Change and Release Management will become history as it will be possible to deploy a new version of an application directly in a small, limited part of the production environment versus moving through the traditional waterfall development pipeline DEV → UT → SIT → UAT → PROD (Development, Unit Test, System Integration Test, User Acceptance Test, Production). The time and cost savings are substantial, and the management of these infrastructures becomes less cumbersome by avoiding a lot of 'red tape'. However, so far we don't see the ability to fully take advantage of the DevOps paradigm in many production environments. The following picture attempts to characterize the key differences between the typical DevOps architectures from companies 'born on the Cloud' versus the traditional JEE architecture on which typically hundreds of our customers' applications run (Fig. 3).

Simply stated, a DevOps-ready application is built on massively parallel scale-out architectures, usually leveraging a (proprietary) map/reduce environment (such as Hadoop, Spark, Cassandra, Hive or the like). First, this means that every node is running a copy of the application, however, they can be on different version levels as there are no interdependencies. Second, the application logic is completely decoupled from the database schema; in fact, since we are mostly dealing with NoSQL data stores based on name/value pairs (with loose consistency between

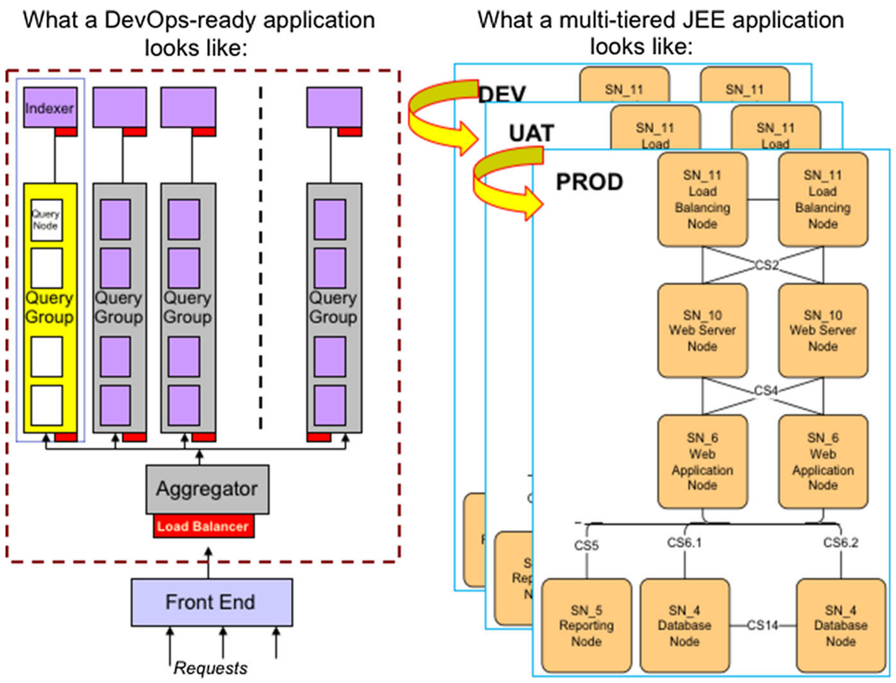


Fig. 3 DevOps-ready application architecture versus legacy JEE application architecture

three replicas vs. ACID in traditional relational database management systems), there is no database schema at all! These two factors make that the overall architecture is resilient to changes in the application in this massive parallel system.

In contrast, today's enterprises still run their applications in a JEE environment, potentially with the help of DevOps deployment tools such as Jenkins, Urbancode Deploy and the like. However, there is no way of deploying a new version of an application e.g., on a single node in a JEE application server cluster while leaving the other nodes run the prior version, let alone without any changes to the underlying database schema (which, in turn can only be either updated for all DB schema copies, or none). As a third differentiator, many enterprises keep emphasizing that they run mission critical applications and are absolutely dependent on transactional integrity. Hence, failure—even of a single node—is not an option for the foreseeable future, and neither is the demise of Change and Release Management. On the flip side, there is hardly a new software migration or re-design project that does not explicitly address the deployment aspects. It is hence fair to say that, over time, the amount of DevOps-ready systems will substantially increase.

2.3 Regulated Industries

As mentioned so far, the typical ITIL use case in enterprise IT management, which drove the creation of integrated service management platforms, consists in accurately representing the managed environment as well as keeping a log of all incidents, problems and changes that have been opened/performed against systems within this environment.

Many service providers have added the need for the above to function seamlessly in a multi-customer environment so that an employee can work across accounts while preventing individual accounts from seeing the data of other accounts. While a fine-grained security model based on role based access control (RBAC) allows the separation of data, contemporary service management platforms have not achieved full multi-tenancy yet.

Regulated Environments—especially in the healthcare, financial and energy sectors—bring additional challenges due to them being subject to very severe audit and compliance requirements as mandated by laws such as the United States Health Insurance Portability and Accountability Act (HIPAA) or standards such as Payment Card Industry Data Security Standard (PCI DSS): the ability of the system to withstand any tampering of logs and audit trails, and the ability to establish non-repudiation of change management approvals and a permanent audit trail of any actions being carried out on the system (both runtime and configuration). While no prevalent solution exists yet, it is conceivable that we will see Blockchain based implementations that address the non-repudiation needs of heavily regulated industries.

2.4 The Elusive Partner Ecosystem

There is no shortage of platforms and frameworks in the Service Management and Automation space. In large companies, a steady flow of acquisitions and business partnerships yields demand for properly integrating a new tool into the already existing ecosystem. This, however, means providing the content for these platforms, such as discovery probes or automation packages for a vast variety of managed resources—and a large variety of versions. However, looking at the history of systems in the space leaves the impression that the buildout of content has been at best a mixed success. There are two reasons for this:

1. Many service management systems share a lack of development budget being allocated to equip the tools with a critical amount of content for the management modules right out of the box. They typically ship with demo code to showcase how this content could be developed, but not a complete suite of modules that would support the most common managed resource types.
2. In addition, the hope of an emerging ecosystem of business partners and open source contributors to build content for all sorts of managed resources often does not materialize, which leaves many tools stranded with a limited footprint, and at risk of being marginalized and ultimately discarded.

An additional burden placed upon these systems is the limited timespan that large software vendors allow for these products to succeed in the marketplace (typically around 2 years); if the commercial success does not happen within this time frame, a product is either withdrawn or blended with another product. This, in turn, leaves the content creators in the situation that similar functionality is being implemented over and over again on ever-changing platforms.

3 Progressing Service Management Automation

This section provides directions on how to advance the agenda for Service Management Automation. It is clear from the foregoing discussion that the way how Service Management is implemented today needs to change so that the efficiency of ITIL can be improved while maintaining its effectiveness. This is needed because mere labor arbitrage (aka ‘offshoring’ or ‘out-tasking’) and financial engineering have run their course and will not yield further savings.

On the other hand, the author has witnessed how naive attempts to ‘remove ITIL’ in pre-sales situations for Hybrid IT (and even pure Clouds) led to quite unpleasant customer reactions, especially in financial or regulated environments. For example, one needs to be careful not to confuse provisioning with change management: Simply ordering a new virtual server versus patching an existing one (and hence avoiding the dreaded change management process) is not an acceptable practice in production environments because most software architectures in use today do not allow this to occur easily (see Sect. 2.2). In addition, the lifecycle of a server begins with the initial order: adding another node to a cluster, stopping/rebooting systems and performing typical lifecycle management operations tend to have an impact on the stability of the overall distributed environment and will require change management.

In the next few sections, we will offer a set of suggestions that we believe will lead to a significant improvement in the Quality of Service Management.

3.1 Event Management: Log Analysis and Machine Learning

We start our discussion with the area of Event Management, which traditionally has been the most advanced in terms of automation, due to—in part—the forced uniformity of the event messages, and the availability of classification and policies, specified as Event-Condition-Action rules. In addition, early successes in pattern detection [8] and event correlation [9] have proven that control-theoretic and machine learning approaches can be successfully applied to Event Management. Log Analysis systems such as Splunk are a contemporary extension of these concepts.

For the same reasons, Automation systems can be deployed in a very effective manner in this area: by focusing on automating the troubleshooting of the most frequent events (whereas 'filesystem $>$ 80% full' is on spot #1), we have seen the reduction of critical issues in server operations by more than 89% along with a decrease of server downtime by more than 50%. The is due to the fact that by automating not only severity 1 tickets, one can catch and remediate misconfigurations of a server at an early stage, long before it becomes a severity 1 issue.

We note that a good practice for maintaining tracebility is to automatically cut an Incident (and potentially a change) ticket whenever Automation performs changes on a managed system. In addition, the generated business value needs to be surfaced: Just because the remediation appeared 'effortless' due to Automation, it still delivered business value and thus should be reported and eventually billed.

3.2 Incident Management: Cognitive Systems, ChatOps

As discussed in Sect. 2.1.3, one of the key issues in Incident Management is the lack of scalability of the submission mechanism and the difficulties of engaging an end user in a conversation or a chat while extracting the necessary information required to submit an incident, instead of presenting the user with a mere 'ticket form'. This is a great opportunity for applying Cognitive Systems such as IBM Watson to Service Desk tasks such as Incident Management.

Studies have shown that the ability for a cognitive system to detect the mood of a submitter is key to guide the conversation, regardless of whether it is happening via phone or text/chat messages: '*The best UI is No UI*' sums up the approach quite nicely. This is an opportunity for deploying cognitive technologies/chat bots with voice-to-text gateways, if needed.

An example of such a conversation is depicted in Fig. 4, where a cognitive system is used as a front-end to a traditional service desk. We obviously still need a data store provided by a ticketing system and a CMDB in order to determine the context for guiding the responses of the cognitive system.

The promise of a cognitive approach lies in 'programming' the system by feeding it a (very) large amount of data versus implementing a rule based system; since service management platforms collect a massive amount of ticket data, the

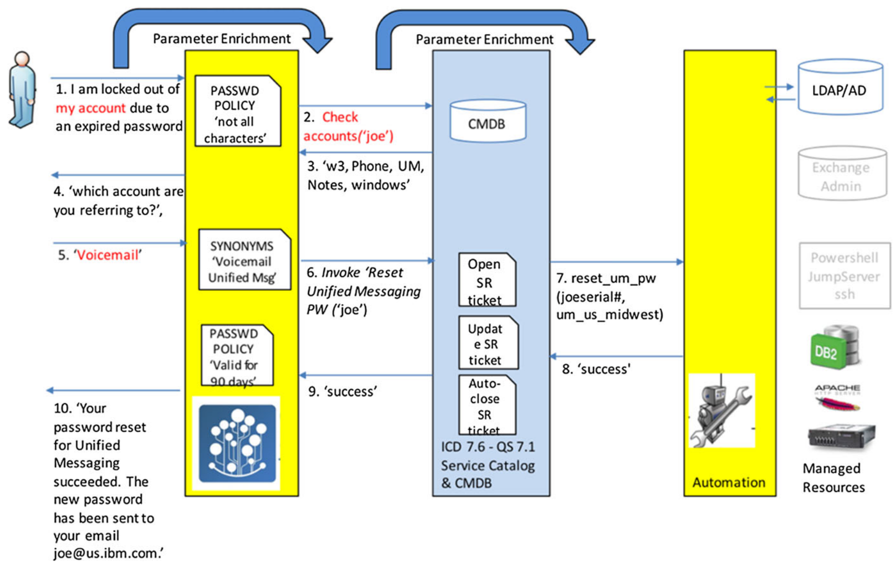

Parameter Enrichment
Parameter Enrichment
1. I am locked out of my account due to an expired password
PASSWD POLICY 'not all characters'
2. Check accounts('joe')
3. 'w3, Phone, UM, Notes, windows'
CMDB
4. 'which account are you referring to?',
5. 'Voicemail'
SYNONYMS 'Voicemail Unified Msg'
6. Invoke 'Reset Unified Messaging PW ('joe')
Open SR ticket
Update SR ticket
Auto-close SR ticket
7. reset_um_pw (joeserial#, um_us_midwest)
8. 'success'
9. 'success'
PASSWD POLICY 'Valid for 90 days'
10. 'Your password reset for Unified Messaging succeeded. The new password has been sent to your email joe@us.ibm.com.'
ICD 7.6 - QS 7.1 Service Catalog & CMDB
Automation
LDAP/AD
Exchange Admin
Powershell JumpServer ssh
DB2
APACHE
Managed Resources

Fig. 4 Using cognitive systems as front-ends for Incident Management

prerequisites for training the system are met. In practice, we find that these systems do not train themselves and need, especially early in the learning cycle, information architects to monitor responses and train the environment.

We note that our backend service desks still cut a ticket for each request, and keep track of it via a reference number. It is just that the system doesn't expose the complexities and rigidity of 'the engine' anymore that performs the work on behalf of a user.

A variation of this concept is known as 'ChatOps', where a cognitive system listens in on group chats or 'channels' as offered by e.g., Slack, and picks up information ('*several users of server x are experiencing poor performance*') to generate incidents for subsequent automated troubleshooting.

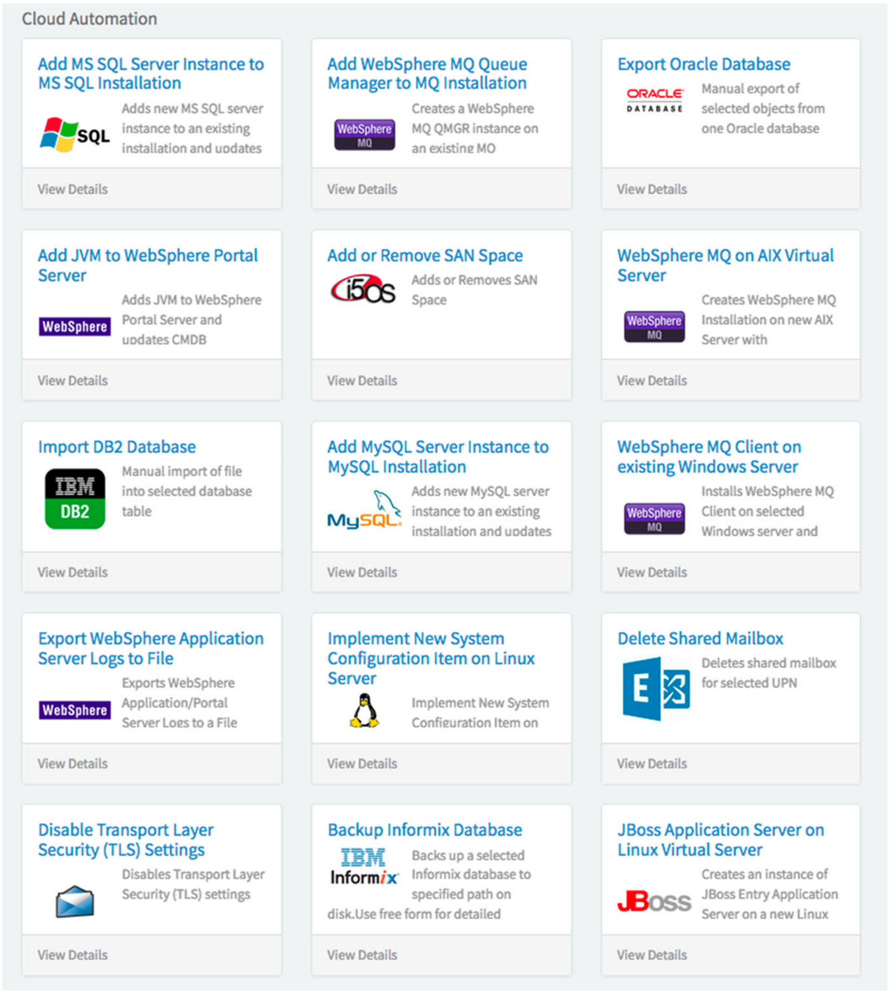
3.3 Service Request and Service Catalog Management

If we extend the above concept of filling out forms through a conversational approach, we increase the domain of use cases to the content of a Service Catalog in a straightforward way. This means that a user may be able to submit a server reboot request (or even placing an order for a new server) verbally or through text messages. This, in turn, requires that enterprises buy into modern concepts of distributed systems such as resource pools, dynamic workload scheduling, auto-scaling and sharing of software licenses versus 'bring your own license key'. The foregoing list is close enough to the definition of a Cloud by NIST [10]:

1. On-demand: Self-Service
2. Elastic: scale up/down dynamically
3. Shared: pooled resources

4. Metered usage: fine level of granularity
5. Accessible via network

It also highlights the fact that a service request catalog as depicted in Fig. 5 is more than a mere assembly of orderable services: If done right, the catalog offers the full spectrum of server and software lifecycle management, from initial provisioning to configuration changes and ultimately decommissioning, while presenting only the systems that are in scope of a specific operation and user, e.g., by segregating systems according to business units, and by greying out menu options depending on their feasibility (a stopped server can only be started, and hence a ‘stop server’ operation in the selection menu is greyed out). The context menus of the service request catalog are driven by data from the CMDB/CMS. Furthermore, a


Cloud Automation

Add MS SQL Server Instance to MS SQL Installation
Adds new MS SQL server instance to an existing installation and updates
View Details

Add WebSphere MQ Queue Manager to MQ Installation
Creates a WebSphere MQ QMGR instance on an existing MQ
View Details

Export Oracle Database
Manual export of selected objects from one Oracle database
View Details

Add JVM to WebSphere Portal Server
Adds JVM to WebSphere Portal Server and updates CMDB
View Details

Add or Remove SAN Space
Adds or Removes SAN Space
View Details

WebSphere MQ on AIX Virtual Server
Creates WebSphere MQ Installation on new AIX Server with
View Details

Import DB2 Database
Manual import of file into selected database table
View Details

Add MySQL Server Instance to MySQL Installation
Adds new MySQL server instance to an existing installation and updates
View Details

WebSphere MQ Client on existing Windows Server
Installs WebSphere MQ Client on selected Windows server and
View Details

Export WebSphere Application Server Logs to File
Exports WebSphere Application/Portal Server Logs to a File
View Details

Implement New System Configuration Item on Linux Server
Implement New System Configuration Item on
View Details

Delete Shared Mailbox
Deletes shared mailbox for selected UPN
View Details

Disable Transport Layer Security (TLS) Settings
Disables Transport Layer Security (TLS) settings
View Details

Backup Informix Database
Backs up a selected Informix database to specified path on disk.Use free form for detailed
View Details

JBoss Application Server on Linux Virtual Server
Creates an instance of JBoss Entry Application Server on a new Linux
View Details

Fig. 5 Service catalog in the ServiceNow platform (extract)

sufficiently rich service request catalog that covers the entire lifecycle of a large set of operating systems and software stacks essentially represents the equivalent of a 'Data Center API' that can be triggered not only by a user through the Service Desk, but also through the API by event management systems to, e.g., initiate autoscaling or kick off remediating actions to address an incident.

The reader will observe that Fig. 5 contains 'atomic' lifecycle management operations, and does not necessarily provide Service Requests that would correspond to an OpenStack HEAT pattern, such as 'deploy a 3-tiered JEE application with a 3-node web application server cluster along with a highly available database cluster'. The reason is that we found that a simple server request for a mere operating system and multiple storage volumes and (virtual) network interface cards usually contains around 200 parameters. A service request for server with a middleware stack easily tops 400 parameters. Obviously, in order for this to work well, the amount of parameters that a user needs to supply must remain fairly small: Seven parameters are typically considered the acceptable maximum by interaction designers because studies have shown that the number of distinct items that a human being can remember concurrently is seven. We found that setting default values in 'Ticket Templates', generating hostnames, assigning IPs from predefined pools and deriving values via CMDB look-ups brings a typical set of 200 order parameters per server down to the following ten mandatory inputs: Business Unit, Securityzone, Environment, OS Version, purpose of request, storage tier, first data disk size, #CPU, #RAM, Business Application Name. Combining the latter 2 (or even 3) parameters into a T-shirt size, gets us within the realm of administrators submitting server builds without complicated interactions on a screen, and potentially via a voice interface that is running on a cognitive system!

While most of these parameters can be defaulted, it would be too cumbersome to list all parameters for an OpenStack HEAT pattern comprising a multitude of servers on the GUI of a Service Management Platform.

We rely instead on a Cloud Broker to allow a user to graphically depict the topology of a distributed system, allow the designation of Development/Test/Production environments, and collect the overall approvals. The Cloud Broker would then create the bill of materials, suggest the most cost-effective service provider, and request each software stack from the service request catalog of the service provider to execute the provisioning actions. Powerful data federation mechanisms from a Cloud Broker to the various service catalogs is needed in order to decouple the Cloud Broker from the actual service request catalogs, in case any changes are made. Keeping the bill of materials in the Cloud Broker enables this system to perform billing as well as chargeback/showback tasks, assuming that it interrogates the authoritative data store. We discuss the approaches for improving the accuracy of the CMDB/CMS in the next section.

3.4 Configuration Management: From Discovery to CMDBs

The aforementioned ITIL best practices demonstrate that managing Hybrid Clouds is a data-intensive endeavor, and the CMDB/CMS is the place where Service

Management platforms store this information in order to become the 'context engine' that ties together users, organizations, cost information, software licenses with managed objects such as clusters, server pools, VLANs, NICs, IP addresses or disks.

The great diversity of the aforementioned managed resources implies that the Management Information Model must be rich. It must have managed objects with properties, typed relationships and the ability to navigate them efficiently. While progressive authors call such an object-oriented data store a 'graph database', the concept of a Common Information Model has been around for 15 years [11].

The question remains what is different now compared to the prior state of the art? The way how CIs are created has changed so that we do not rely on discoveries anymore!

In particular, the major measures to addressing the capability gaps identified in Sect. 2.1.2 are:

- we federate data that relates to users, locations or organizations (from LDAP servers),
- pre-board cost centers, and
- rely on Cloud platforms for placement of VMs into VLANs and the selection of IP addresses.

All of this information is gradually filled in during the provisioning process and the service request ticket evolves into a build sheet as provisioning progresses. Upon success of the provisioning process, we automatically generate(!) the Authorized CIs in the CMDB/CMS from the ticket data. There are about 15-20 CIs per Server with OS installed and about 30-35 for a Server with Middleware/Database.

As a consequence, discovery technologies such as Chef Ohai [12] may come into play as a secondary control, namely in order to perform automated CI audits. Every once in a while, a discovery scan of the environment is performed in order to identify discrepancies between the authorized ('what it should be') and the actual ('what it really is') state of the CIs. Every discrepancy (more CPU, RAM or disk space than initially ordered) may be a hint that unauthorized changes are being done, and hence trigger an incident for subsequent reconciliation and—eventually—investigation. We found that analyzing the incidents also yields valuable information on finding errors within the automation itself, or with the integrity of the data that was used for establishing the networks, VLANs and security zones within the environment.

The key insight here is that the aforementioned allow to drastically cut down the time it takes to obtain current CI information, and hence circumventing the long-winded and inevitably incomplete discovery procedure. The very same mechanism works fine for all subsequent lifecycle management operations (modify system parameters), including decommissioning. The latter is done by merely archiving the CIs so that billing can be done based on actual time used, even if the server has been decommissioned prior to the billing report.

3.5 Change Management: Leverage Standard, Pre-approved Changes

Once a service request has been submitted through the catalog, its execution will in most cases require a change, especially in a production environment. We can cut down the complexity of change management substantially by reminding ourselves that ITIL has long ago introduced the concept of standard, pre-approved changes which—by definition—do not require Change Advisory Board (CAB) approvals. These are changes whose implementation carries no or only very minor risks, and whose execution is predictable, especially if done through automation. We can therefore classify the typical set of canonical lifecycle operations into 3 categories:

1. Standard, pre-approved changes.
2. Normal changes, potentially requiring a CAB.
3. Changes that are rare or require expert knowledge, and which should probably not be in a service catalog at all, but should rather be done as a separate, dedicated project.

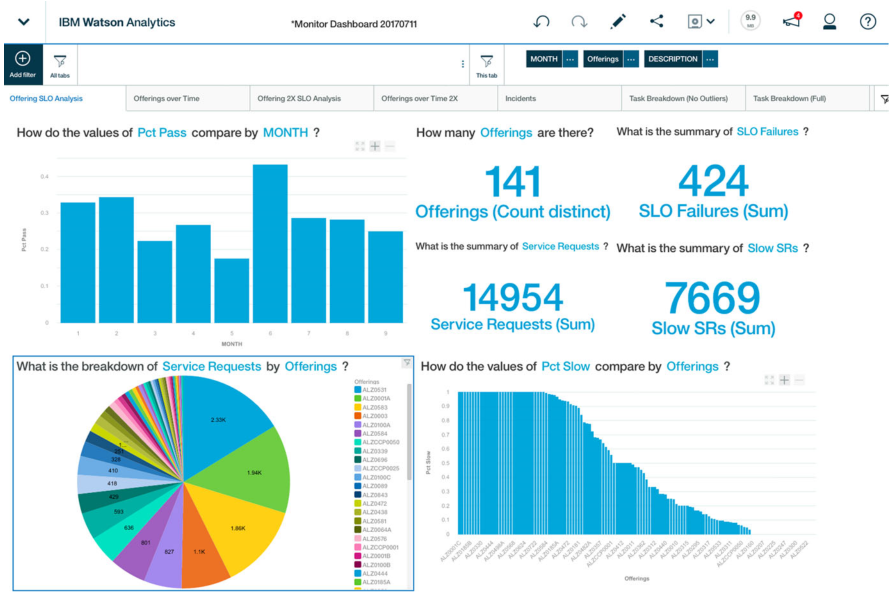
When it comes to classifying changes into the above 3 categories, we have witnessed that the opinions of our clients differ greatly. While some are more ‘lenient’ with respect to checks and approvals, others lack trust on whether a distributed resource scheduler will place a VM onto a server with enough remaining capacity and whether the system is able to determine the right amount of thin storage provisioning without running out of disk space. Thus, they demand a normal change even for provisioning tasks.

We fully agree with Forrester [7] that the goal has to be to ‘*lower the center of gravity*’ by allowing a large portion of standard, pre-approved changes. We also expect emerging software architectures based on Spark/Hadoop for scale-out systems (cf. Sect. 2.2) to gradually eliminate the single points of failure that until now require human impact analysis and sign-off on changes that could negatively affect critical parts of the infrastructure.

3.6 From Reporting to Analytics

We end our discussion with the Reporting and Dashboard Building Block, introduced at the beginning of Sect. 2. At the core of Service Management Reporting lies Service Level Management, which ties the key performance indicators of a service provider to dollars. Our Research work, initially described in the present journal 14 years ago [13] has matured over time into a solution that IBM uses on commercial accounts for measuring SLA attainment. Figure 6 illustrates a sample dashboard that is built on the common-off-the-shelf Business Intelligence tool IBM Watson Analytics, and which is fed by the SLA engine that was first described in [13].

Figure 6 depicts SLA attainment for a total of 9 months of data (left upper corner) for an account that uses our Service Catalog described in Sect. 3.3, and whose usage of service request types is depicted in the pie chart in the lower left corner. The dashboards are dynamic in the way that the individual performance of a


IBM Watson Analytics

*Monitor Dashboard 20170711

Offering SLO Analysis | Offerings over Time | Offering 2X SLO Analysis | Offerings over Time 2X | Incidents | Task Breakdown (No Outliers) | Task Breakdown (Full)

How do the values of Pct Pass compare by MONTH ?

How many Offerings are there? 141 Offerings (Count distinct)

What is the summary of SLO Failures ? 424 SLO Failures (Sum)

What is the summary of Service Requests ? 14954 Service Requests (Sum)

What is the summary of Slow SRs ? 7669 Slow SRs (Sum)

What is the breakdown of Service Requests by Offerings ?

How do the values of Pct Slow compare by Offerings ?

Fig. 6 Service level agreement dashboard (sample)

given service offering can be measured, as well as the different tasks in the delivery processes. Since each task has an owner (potentially a different service provider as we are dealing in a multi-sourcing context), we are then able to establish a rating of the different providers and teams that contribute to the delivery of a service. Additional comparisons by geography and business unit are doable as well, given that slightly different delivery processes are in place.

4 Conclusions and Outlook

In this paper, we have described the key challenges and achievements of Network, Systems and Service Management that we have witnessed over the past 25 years. The Journal of Network and Systems Management has been the forum where a good deal of the innovation of our cherished discipline has been described. While we are pretty sure that self-driving cars will have become a reality 25 years from now, in 2042, it remains to be seen if the Automation of Service Management tasks will proceed at an equally brisk pace, and if Service Management as an industry will exist in 25 years at all, and in which shape.

There is a case to be made that continuous standardization on open source technologies coupled with cognitive systems and Cloud automation tools will make that Service Management will disappear as a distinct technology and 'weave itself into the fabric of everyday life' [14] in order to become a 'most profound technology'.

The authors and readers of the Journal of Network and Systems Management will play a major role in shaping the future of this important discipline.

References

1. Krol, E.: The Whole Internet, 1st edn. O'Reilly & Associates Inc., Sebastopol (1992)
2. Waze: Get the best route, every day, with real-time help from other drivers. https://www.waze.com. Accessed July 2017
3. Baker & McKenzie's Global Sourcing Practice Group: Introduction to Outsourcing. Baker & McKenzie (2008)
4. Karamouzis, F., Da Rold, C.: Predicts 2014: Business and IT Services Are Facing the End of Outsourcing as We Know It. Gartner Inc, Stamford (2014)
5. Axelos: ITIL Service Transition. AXELOS, London (2011)
6. Amazon: Amazon web services cloud products. https://aws.amazon.com/products/ (2017)
7. Betz, C.: Change Management: Let's get back to Basics. Forrester Research, Cambridge (2017)
8. Ma, S., Hellerstein, J.: Mining partially periodic event patterns with unknown periods. In: Proceedings of the 17th International Conference on Data Engineering, Washington, DC (2001)
9. Jakobson, G., Weissman, M.: Real-time telecommunication network management: extending event correlation with temporal constraints. In: Integrated Network Management IV: Proceedings of the fourth international Conference on Integrated Network Management, Toulouse, France (1995)
10. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. National Institute of Standards and Technology, Gaithersburg (2011)
11. Distributed Management Task Force: Common Information Model http://www.dmtf.org/standards/cim (2017)
12. CHEF Ohai: About Ohai. https://docs.chef.io/ohai.html. Accessed July 2017
13. Keller, A., Ludwig, H.: The WSLA framework: specifying and monitoring service level agreements for web services. J. Netw. Syst. Manag. **11**(1), 57–82 (2003)
14. Weiser, M.: The computer for the 21st century. Sci. Am. **265**(3), 94–104 (1991)

Alexander Keller is an IBM Distinguished Engineer and Director, Integrated Service Management with IBM Global Technology Services in Chicago, IL, USA. In this role, he is responsible for setting the technical strategy on a global level and works with many customers on complex Cloud and IT Service Management implementation projects. Alexander's core areas of expertise are large-scale Discovery systems, Service Desks, Change and Configuration Management implementations, and Cloud Computing. Prior to joining IBM's Global Services organization in 2007, he managed the Service Delivery Technologies department at the IBM T.J. Watson Research Center in Yorktown Heights, NY. He received his M.Sc. and Ph.D. degrees in Computer Science from Technische Universität München, Germany, in 1994 and 1998, respectively and has published more than 60 refereed papers in the area of distributed systems and IT service management.